Formalna verifikacija programske potpore: Uvod

Sveučilište u Zagrebu

Fakultet elektrotehnike i računarstva

Pripremio: izv. prof. dr. sc. Alan Jović

Ak. god. 2022./2023.

Što su formalne metode (FM)?

- To su **matematički zasnovane tehnike** za specificiranje zahtjeva i arhitekture u oblikovanju i razvoju, te za verifikaciju sklopovskih i programskih sustava

- Uporaba FM je motivirana istim argumentima kao i u drugim inženjerskim disciplinama: matematički zasnovana analiza **doprinosi pouzdanosti i robusnosti konačnog proizvoda**

- FM nisu zamjena za ispitivanje (testiranje) već su to dva međusobno **komplementarna** skupa tehnika
  - Ispitivanje – dinamička verifikacija i formalna verfikacija (dio FM) se nadopunjuju
  - U FM usredotočenost je češće na **statičko** (simboličko) rasuđivanje o sustavima

Što je zajedničko većini formalnih metoda?

- Jezik
  - FM su matematički način zapisa ili računalni jezik s formalnom semantikom koji omogućuje opis svojstava sustava i svojstava koja se od sustava očekuju. Sustav je opisan kao matematički objekt na rigorozan i analitički način putem nekog jezika
- Alat
  - FM dolaze u obliku računalnih alata koji osiguravaju da se sustav izvede i u konstrukcijskom i u funkcionalnom smislu prema očekivanjima korisnika
- Metodologija
  - Da bi bili učinkovite, FM moraju biti dobro integrirane u industrijsku praksu, stoga većina formalnih metoda ima smjernice za ispravnu uporabu u razvoju stvarnih sustava

Formalne metode u primjeni

- Primjena u životnom ciklusu (engl. *life-cycle*) razvoja sustava
  - u svim fazama ciklusa razvoja
  - samo u nekim fazama razvoja, često u fazi validacije i verifikacije

- Primjena na komponente sustava
  - na sve komponente
  - samo na neke, i to one složenije i kritične (češće)

- Primjena na svojstva (engl. *properties*) komponente
  - primjena na sva funkcijska svojstva (puna funkcionalnost)
  - primjena samo na neka svojstva, obično na ona kritična

Formalne metode u industrijskoj praksi

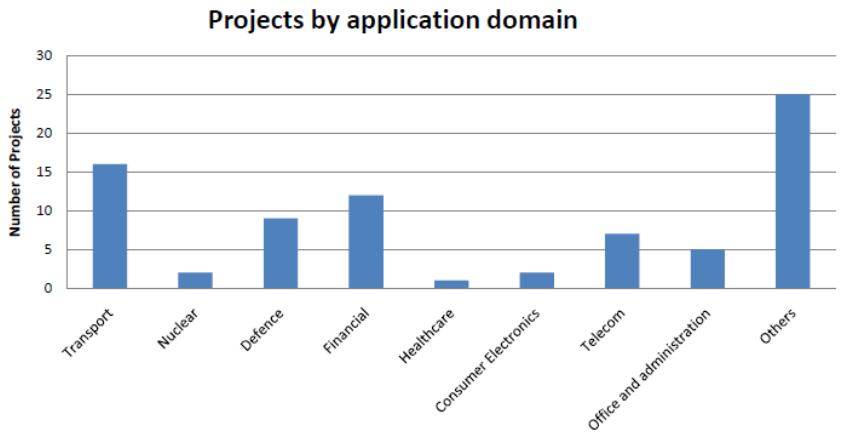
FM se najčešće koriste za sustave gdje je cijena zatajenja visoka

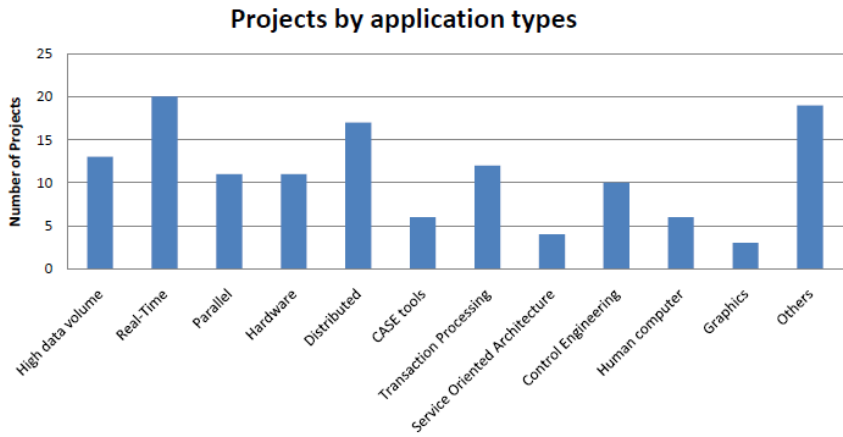
- **Za sustave s kritično važnim ciljem (engl. *mission-critical system*)**
  - Zatajenja su ovdje posebno skupa, a pogreške teško otklonjive nakon puštanja u pogon; najčešće se verificira sklopovlje
  - Npr. navigacija u raketnoj industriji, vojna oprema, proizvodnja čipova
- **Za „po život kritične" sustave (*engl. life-critical system, safety critical system*)**
  - FM su zakonski zahtijevane po tehničkim normama ili certifikacijskim autoritetima; verificira se i programska potpora i sklopovlje
  - Npr. civilno zrakoplovstvo, zdravstvo, željeznice, nuklearne elektrane, roboti
- **Za informacijske i poslovne sustave s visokom razinom sigurnosti (engl. *high-security system*)**
  - Često zahtijevaju rad u skladu s normama i razinama zaštite; verificira se najčešće programska potpora
  - Npr. pristup tajnim državnim podacima, bankovni sustavi

Formalne metode u industrijskoj praksi

- Postoje dokazi da uporaba FM povećava kvalitetu, **smanjuje cijenu i skraćuje vrijeme** stavljanja proizvoda na tržište (engl. *time-to-market*)
- Npr. J. Woodcock, P. G. Larsen, J. Bicarregui, and J. S. Fitzgerald. Formal Methods: Practice and Experience. *ACM Computing Surveys, 41(4), 2009.*

  62 industrijska projekta:
  - Kvaliteta: 92% poboljšanje, 8% nema učinka
  - Vrijeme: 35% poboljšanje, 53% nema učinka, 12% pogoršanje
  - Cijena: 37% poboljšanje, 56% nema učinka, 7% pogoršanje
  - Većina sudionika u studiji složila se da je korištenje formalnih metoda bilo uspješno (jako suglasni: 61%, suglasni: 34%, miješano mišljenje: 5%).



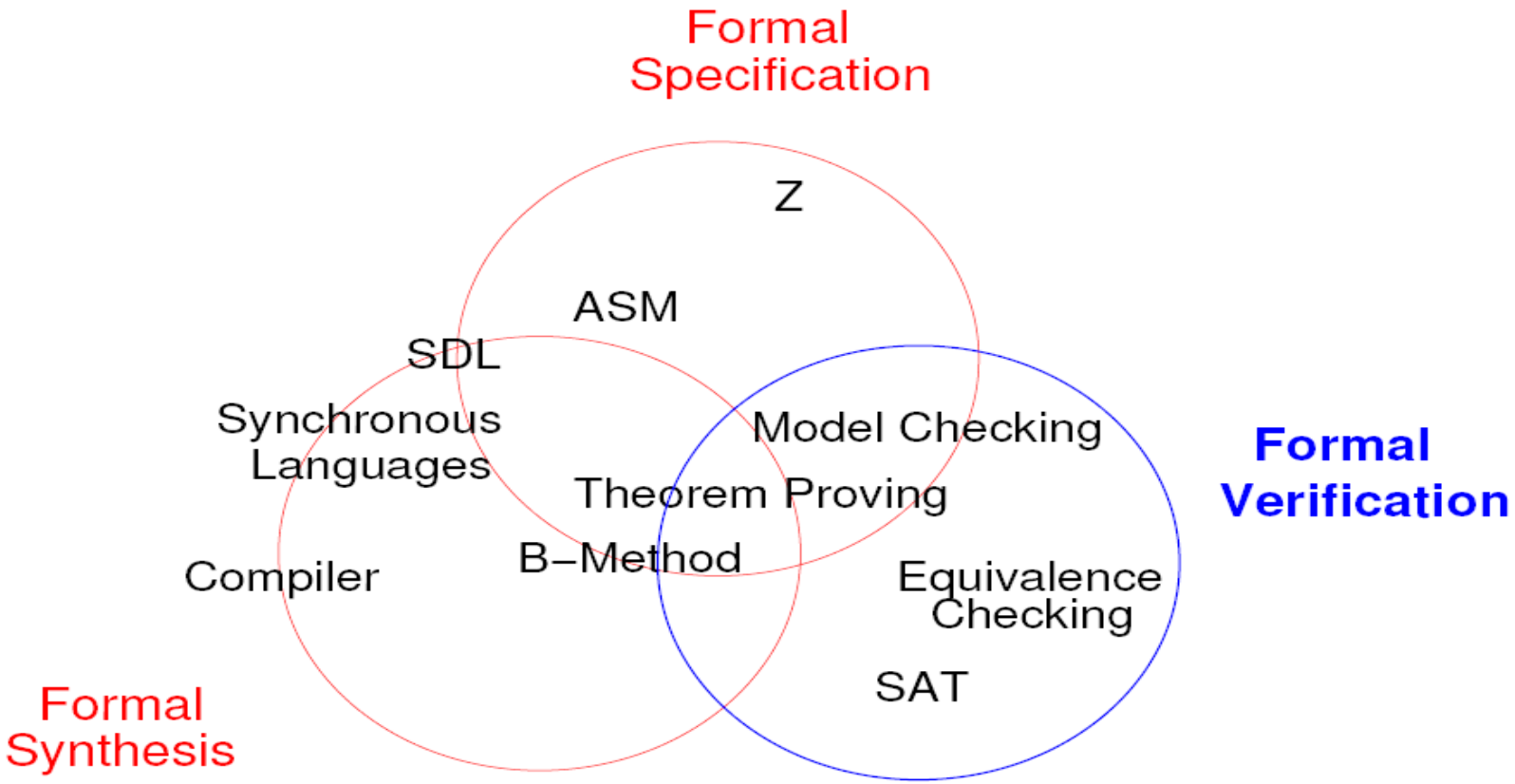


Što se sve može tvrditi o FM?

- Pozitivno:
  - FM su korisne u ranom otkrivanju i eliminaciji pogrešaka
  - FM mogu klijentima bolje pomoći u razumijevanju toga što točno kupuju
  - FM rade tako da vas prisile da puno razmišljate o sustavu koji planirate izgraditi prije kodiranja
  - FM mogu smanjiti cijenu i trajanje razvoja projekta
- Negativno
  - FM u većini slučajeva ne mogu garantirati da je sustav bez pogrešaka – čak i ako je sama FM savršena, čovjek koji ju koristi nije
  - FM su teške za naučiti i primijeniti u praksi
  - Za uspješnu primjenu, često se treba kombinirati više FM

Klasifikacija formalnih metoda

(izvor: Daniel Kroening, University of Oxford, UK)



Formalna specifikacija

- Apstrahira nepotrebne implementacijske detalje.
- To je matematički model sustava koji želimo razviti visoke razine apstrakcije.
- Koristan pristup u početnom oblikovanju sustava.
- Prokazuje **nekonzistentne** i **dvosmislene** specifikacije (koje izviru iz analize zahtjeva).
- **Nema dodatnih alata** – formalna specifikacija je sama za sebe tehnika.
- U nekim slučajevima, formalna specifikacija je i jedina FM koja se koristi u industrijskim projektima

Primjer formalne specifikacije: metoda Z (čita se "zed")

- Metoda je razvijena na University of Oxford, UK (originalno Abrial 1977.)
  "Why Z?" Abrial – "Because it's the ultimate language!" ☺
- Temeljena na logici predikata prvoga reda, teoriji skupova i lambda računu.
- **Z dokument** se sastoji iz matematičkih definicija isprepletenih s engleskim tekstom.
- **Z tekst je pisan kao skup struktura – Schema**.

Značajke

- Započinje se s osnovnim definicijama i ograničenjima sustava.
- Nastavlja se s opisom akcija sustava.
- Završava se dokazivanjem različitih teorema koji garantiraju konzistentnost definicija.
- U deklarativni dio Scheme mogu se uključiti imena drugih postojećih Schema (nasljeđivanje).

Primjer formalne specifikacije: Z metoda

- **Tipovi schema:**
  - State schema – globalne izjave o sustavu.
  - Operation schema – opisuje učinak određenih operacija koje mijenjaju stanje podataka u sustavu.
  - Observation schema – opisuje dohvat informacija (podaci u sustavu se ne mijenjaju).

  - Dozvoljene su sve logičke operacije između Schema kao i kvantifikacija (egzistencijska i univerzalna).

Primjer formalne specifikacije: Z metoda

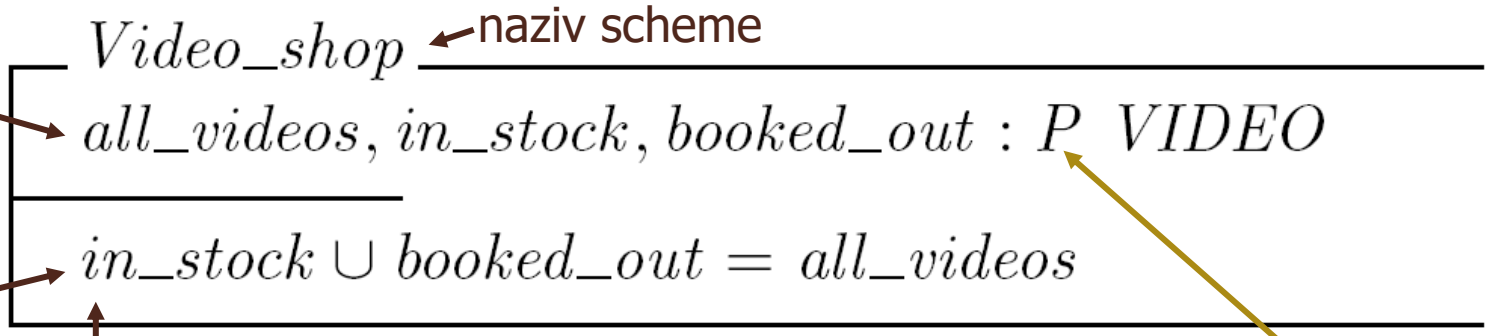
Primjer videoteke (izvor: R.Qu, Univ. Of Nottingham, UK).

- Postoji skup objekata (varijabla) *all_videos* tipa VIDEO.
- Podskup tih objekata je (varijabla) *in_stock* koji se nalaze u videoteci.
- Postoji podskup iznajmljenih videa (varijabla) *booked_out*.
- Pojedinačni video koji posjeduje videoteka mora biti u jednom od dva naprijed navedena skupa (*in_stock* ili *booked_out*).

Primjer formalne specifikacije: Z metoda

State Schema:

$[VIDEO]$ ← osnovni tip (skup) objekata



deklaracije varijabli

odnosi između vrijednosti varijabli

svi podskupovi "powerset"

Može sadržavati i druge odnose kao npr.

$max\_videos : N$

$\#all\_videos \leq max\_videos$

Primjer formalne specifikacije: Z metoda

Operation Schema (npr.: vraćanje videa)

Video treba maknuti iz skupa *booked_out* i dodati u skup *in_stock*.

Video ne smije prije toga biti u skupu *in_stock*.

Uključi schemu ***Video_shop*** uz promjenu stanja Δ

$$
\begin{array}{|l}
\hline
Video\_returned \\
\Delta Video\_shop \\
video? : VIDEO \\
\hline
video? \in booked\_out \\
booked\_out' = booked\_out - \{video?\} \\
in\_stock' = in\_stock \cup \{video?\} \\
all\_videos' = all\_videos \\
\hline
\end{array}
$$

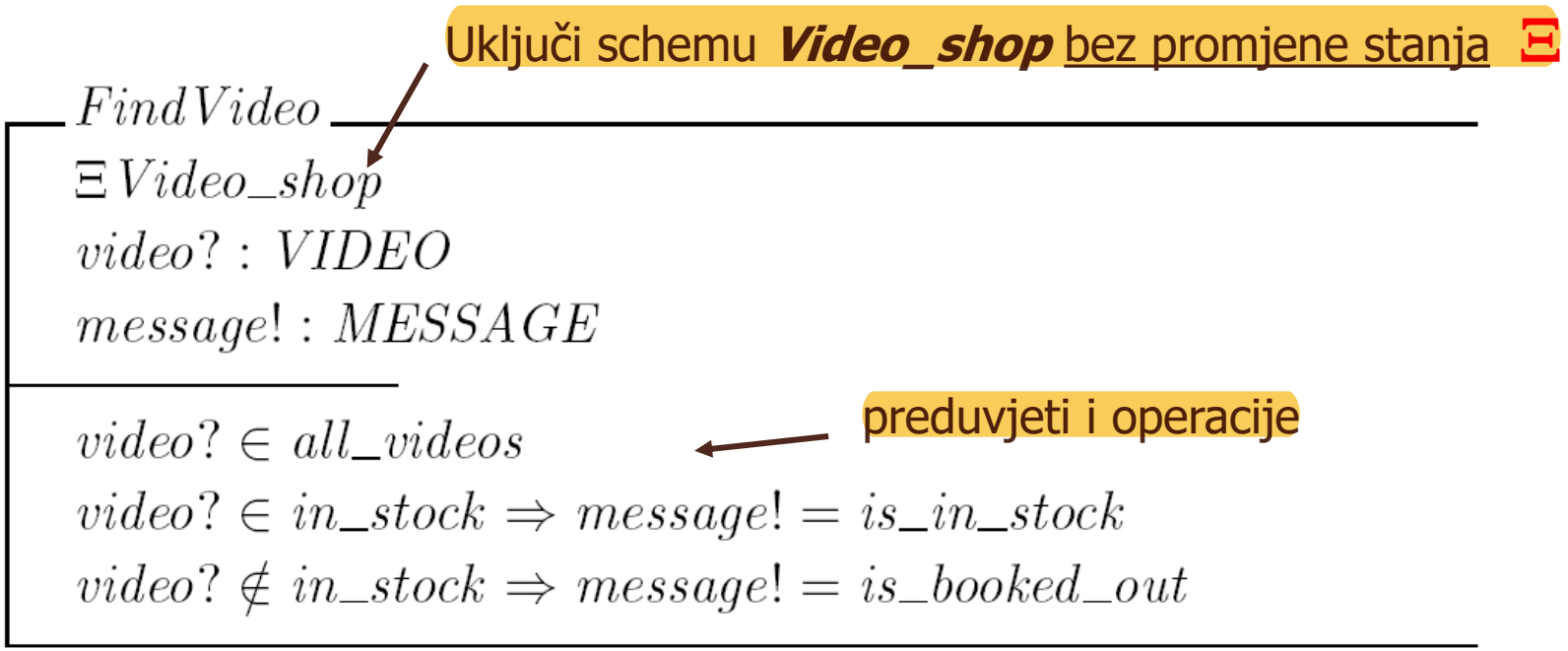
preduvjet koji mora biti zadovoljen prije navedenih operacija

? = ulazne vrijednosti, ´ = nove vrijednosti

Primjer formalne specifikacije: Z metoda

Observation Schema (npr.: je li video slobodan za posudbu)

MESSAGE :: is_in_stock | is_booked_out

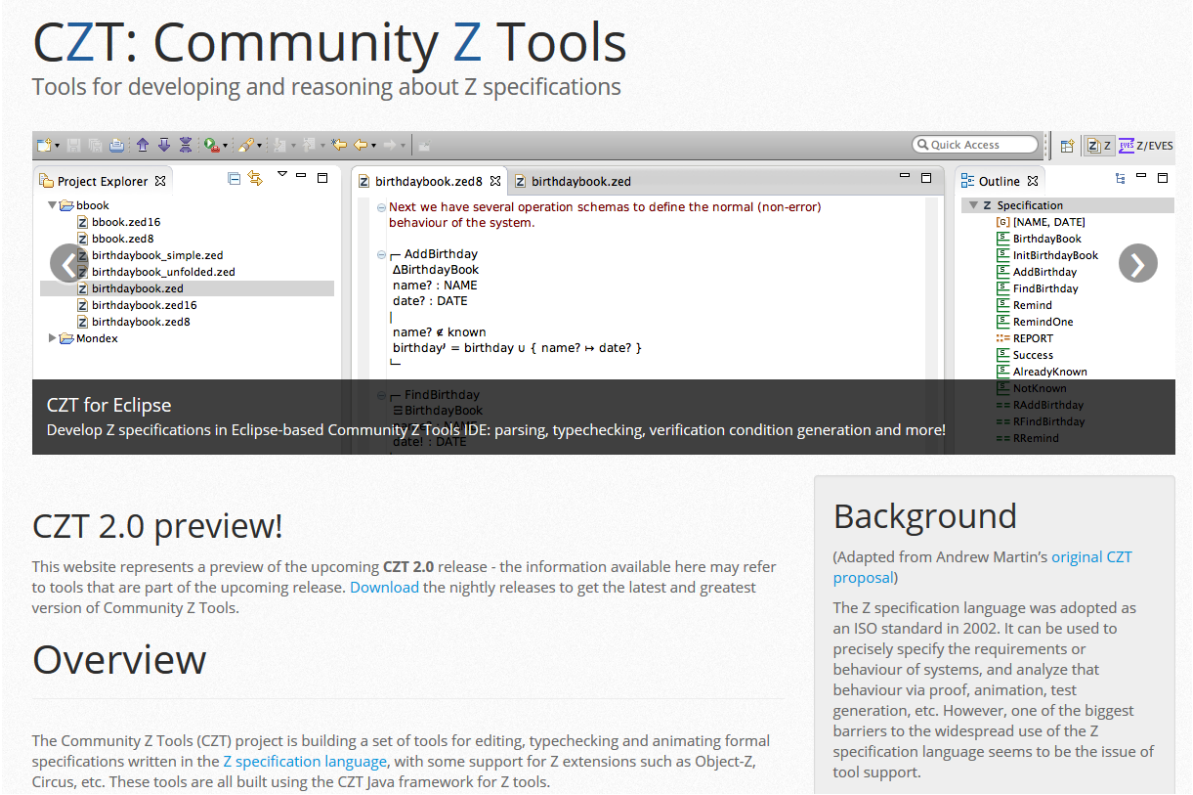


Primjer formalne specifikacije: Z metoda

http://czt.sourceforge.net/

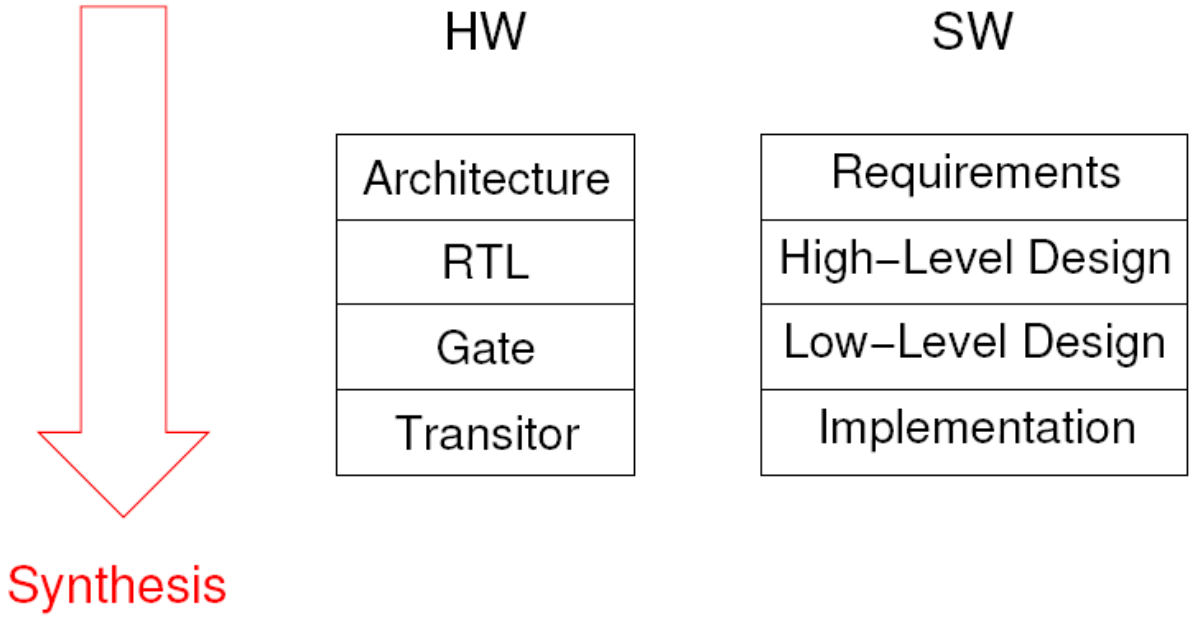
https://sourceforge.net/projects/zwordtools/

https://abz2023.loria.fr/



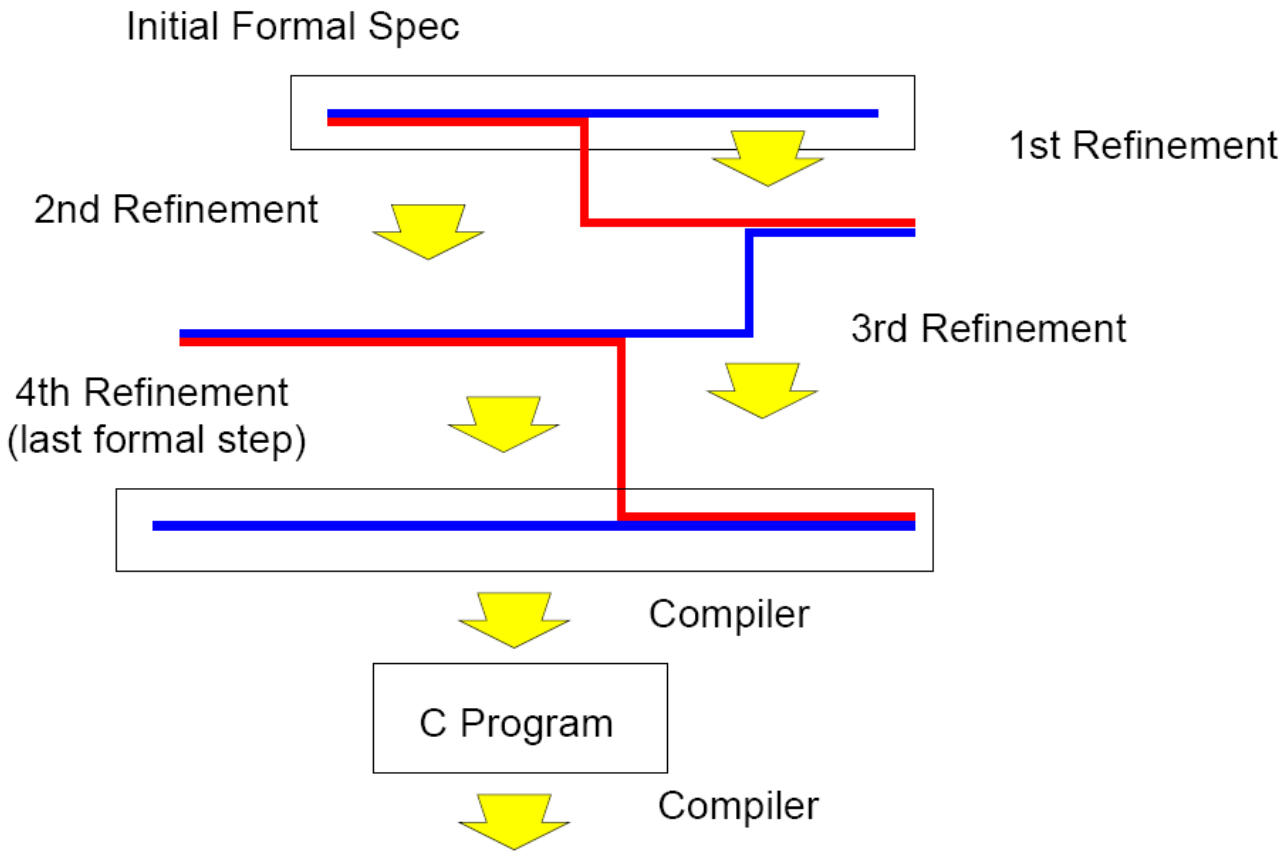
Formalna sinteza

Oblikovanje sustava odozgo prema dolje – od najveće razine apstrakcije do implementacije (rafiniranje), korištenjem formalizama za verifikaciju, pri čemu se raspodjeljuje verifikacijski posao na male zadatke



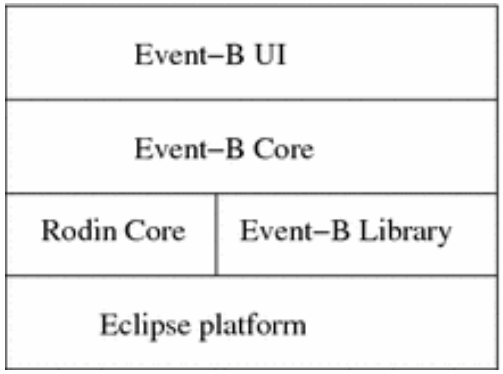
Formalna sinteza

- Ideja:



Primjer formalne sinteze – B-metoda

- **B-metoda i Event-B** (+ platforma **Rodin**)
  - http://www.event-b.org/index.html
- Razvijaju se od 1988. do danas
- Event-B je novija od B-metode i danas se više koristi zbog manje složenosti notacije
- **Event-B** koristi:
  - **Teoriju skupova** kao notaciju za modeliranje
  - **Rafiniranje** za predstavljanje sustava na različitim razinama apstrakcije
  - **Matematičke dokaze** za provjeru konzistentnosti između različitih razina kao način deduktivnog rasuđivanja
- Široko korištena notacija i platforma u industrijskim i istraživačkim projektima: *http://wiki.event-b.org/index.php/Industrial_Projects*



Primjer formalne sinteze – B-metoda

- Svaki **model** u Event-B se sastoji od statičkog dijela – **konteksta** i dinamičkog dijela – **stroja**
- Kontekst sadrži:
  - **konstante** i
  - **aksiome** koji opisuju svojstva konstanti
- Stroj se sastoji od:
  - **stanja** – opisana **varijablama**
  - **događaja** koji specificiraju evoluciju stanja
- Notacija uključuje više ključnih riječi (npr. *when*, *then*, *end*...) koje su samo delimiteri da bi tekstualna reprezentacija modela bila čitljiva
- Više o notaciji:

  http://www3.hhu.de/stups/handbook/rodin/current/html/mathematical_notation_introduction.html

Primjer formalne sinteze – B-metoda

- **Varijable** su jednostavni matematički objekti koji mogu biti
  - Skupovi
  - Binarne relacije
  - Funkcije
  - Brojevi
- Varijable mogu biti ograničene **invarijantama** – logičkim tvrdnjama koje moraju vrijediti pri svakoj promjeni vrijednosti varijable

Primjer formalne sinteze – B-metoda

- **Događaji** se sastoje od **preduvjeta** (engl. *guard*) i od **akcije** (engl. *action*), a mogu imati i **parametre**
- Preduvjet nužno treba biti ispunjen da bi se događaj pokrenuo
- Akcija određuje način na koji varijable stanja evoluiraju pri pojavi događaja
  - Definira skup pridruživanja koja modificiraju stanje
- Samo jedan događaj može biti izabran za izvođenje u nekom trenutku – izbor je nedeterministički

Primjer formalne sinteze – B-metoda

Primjer izrade sustava za provjeru registriranih korisnika pri ulasku i izlasku iz zgrade

1. Stvorimo novi kontekst i dodamo skup objekata *USER* u taj kontekst
2. Stvorimo novi stroj i dodamo varijablu *register* u taj stroj, koja predstavlja skup registriranih korisnika
3. Dodamo invarijantu da bi *register* definirali kao skup korisnika:

   inv1 $register \subseteq USER$

   Automatski, tip varijable *register* je **P**(*USER*) – partitivni skup od *USER*
4. Dodamo događaj `Register` za dodavanje novog korisnika u *register*:

   `Register` $\triangleq$ **any** *u* **where**

   grd1 $u \in USER \setminus register$ // preduvjet: *u* je u *USER* ali ne u *register*

   then

   act1 $register := register \cup \{u\}$ // akcija: *register* je pojačan s novim članom

   end

Primjer formalne sinteze – B-metoda

5. Dodamo događaj inicijalizacije `Initialization`: *register* := ∅
6. Dodamo varijable ljudi koji su u zgradi (*in*) i izvan zgrade (*out*) i dodamo ograničenja:

   inv2 *in* ⊆ *register* // primijetiti da je *register* varijabla

   inv3 *out* ⊆ *register*
7. Dodamo razumljivu invarijantu da korisnik ne može biti istovremeno u zgradi i izvan nje:

   inv4 *in* ∩ *out* = ∅

Ovakav model sada iziskuje šest **matematičkih dokaza** (engl. *proof obligations*), tri se tiču toga da inicijalizacija ustanovljava inv2 – inv4, a tri se tiču toga da događaj `Register` održava invarijante inv 2 – inv4; svi dokazi prolaze

Primjer formalne sinteze – B-metoda

8. Sada trebamo dodati događaje za ulazak u zgradu i izlazak iz nje. Ako pokušamo ovako za ulazni događaj:

`Enter` ≙ **any** *u* **where**
grd1 $u \in out$
then
act1 $in := in \cup \{u\}$
end

Onda dokaz pada na invarijanti inv4 $in \cap out = \emptyset$

Hyp1 : $in \cap out = \emptyset$
Hyp2 : $u \in out$
|–
Goal : $(in \cup \{u\}) \cap out = \emptyset$

Greška je ili u postavljenoj invarijanti ili u događaju `Enter`. Očito, u ovom slučaju je greška u događaju `Enter`, tako da:

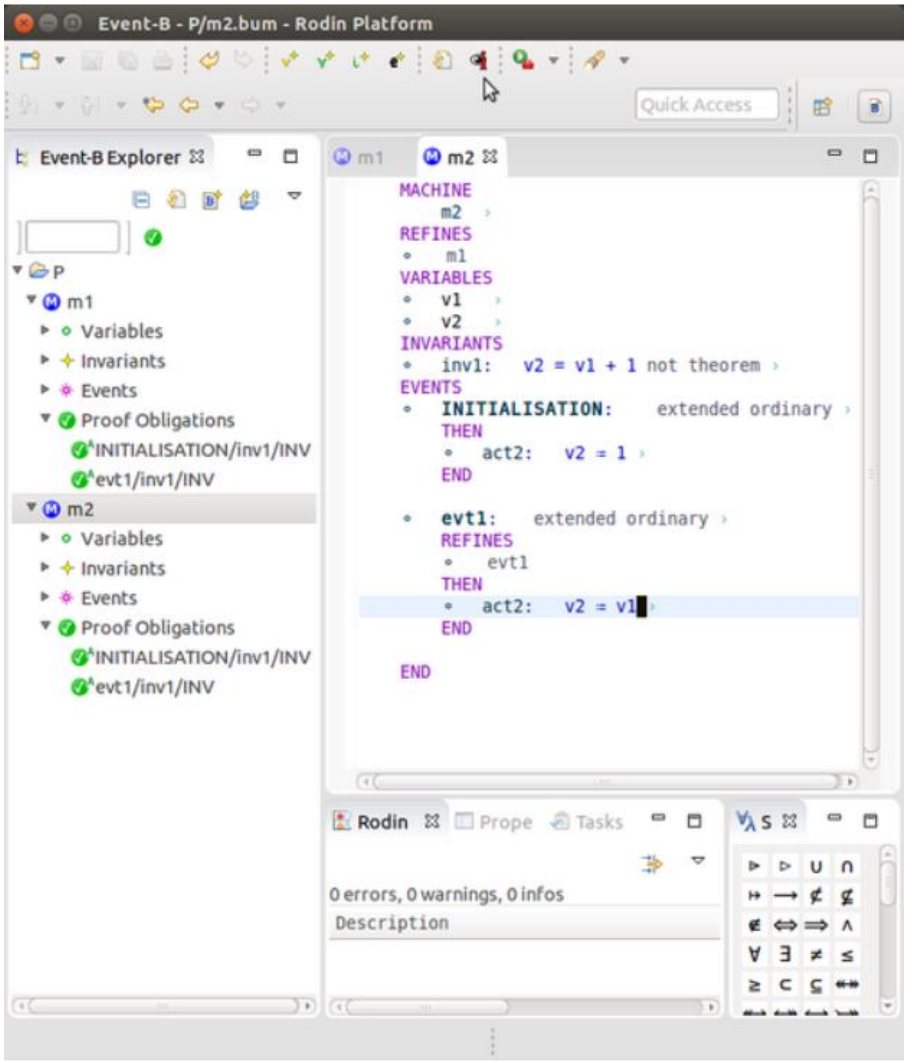
9. Dodajemo novu akciju događaju `Enter`:

act2 $out := out \setminus \{u\}$

i rafiniranje modela ide dalje...

Primjer formalne sinteze – B-metoda

- Platforma Rodin izgrađena je na Eclipse-IDE



Primjer formalne sinteze –

Prevoditelj (engl. *compiler*), Izvor: A. Artale, Univ. of Bolzano, Italy

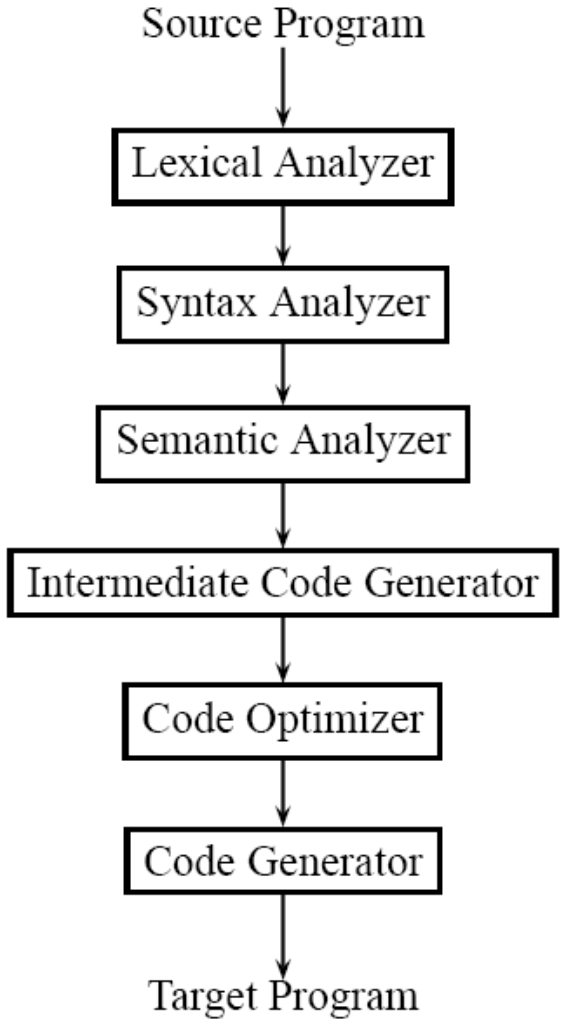
- Prevoditelji preslikavaju programe iz jednog jezika (izvorni jezik) u drugi jezik (ciljni jezik)
- Dodatno, prevoditelju su nužni drugi programi za generiranje strojnog, izvršnog koda
- Interpreteri tumače i izvode program bez njegovog prethodnog prevođenja u strojni kod

Slijedni proces prevođenja:

Izvorni program (*engl. source code*) -> Prevoditelj (engl. *Compiler*) -> Ciljni asemblerski program -> Asembler -> Relokatibilni strojni kod -> Povezivanje (engl. *Linker*) -> Izvršni kod -> Punjenje (engl. *Loader*) -> Završno razrješenje adresa i program pokrenut u memoriji

Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

- Arhitektura prevoditelja
  - Analiza (dekompozicija izvornog programa u dijelove i kreiranje prijelazne reprezentacije).
  - Sinteza (generiranje ciljnog programa temeljem prijelazne reprezentacije).
- Analiza ima sljedeće faze:
  - Leksička analiza
  - Sintaksna analiza
  - Semantička analiza
- Sinteza ima sljedeće faze:
  - Generiranje prijelaznog koda
  - Optimizacija koda
  - Generiranje ciljnog koda



Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

- Leksička analiza i izgradnja tablice simbola
  - Izdvajanje slijeda znakova (leksema) i grupiranje u leksičke značke (engl. *tokens*) određenog tipa.

Npr:

```
position = initial + rate * 60
```

ID - Identifier

Lexeme	Token
position	ID
=	=
initial	ID
+	+
rate	ID
*	*
60	NUM

Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

- Sintaksna analiza = parsiranje (grupiranje znački u gramatičke fraze predstavljene stablom parsiranja).
- Sintaksna analiza daje hijerarhijsku strukturu izvornog programa izraženu preko rekurzivnih pravila ili produkcija (engl. *productions*).
- Semantička analiza provjerava konzistentnost tipa operanda.
- Primjer sintaksnih pravila za naredbe pridruživanja:

$$
\begin{aligned}
< assignment > &\rightarrow \text{ID } \text{“} = \text{”} < expr > \\
< expr > &\rightarrow \text{ID} \mid \text{NUM} \mid < expr >< op >< expr > \mid (< expr >) \\
< op > &\rightarrow + \mid - \mid * \mid /
\end{aligned}
$$

Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

- Generiranje prijelaznog koda (međukoda).
  - Najčešće slijed naredbi s najviše tri operanda takav da:
    - Postoji najviše **jedan operator** pored pridruživanja
    - Generiraju se privremena imena za izračunavanje prijelaznih operacija

- Primjer: `position = initial + rate * 60`
  `id1 id2 id3`

```
temp1 = inttoreal(60)
temp2 = id3 * temp1
temp3 = id2 + temp2
id1 = temp3
```

optimizacija →

```
temp1 = id3 * 60.0
id1 = id2 + temp1
```

Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

- Generiranje (asemblerskog) koda
  - Odabir memorijskih lokacija za varijable (id1, id2, id3).
  - Naredbe višeg jezika preslikavaju se u sekvence asemblerskih naredbi.
  - Varijable i međurezultati pridjeljuju se memorijskim lokacijama.
- Primjer: `position = initial + rate * 60`

`id1` `id2` `id3`

```
MOVF id3, R2        //F=floating_point,R=register
MULF #60.0, R2      // result in R2
MOVF id2, R1
ADDF R2, R1         // result in R1
MOVF R1, id1
```

Primjer formalne sinteze – Prevoditelj (engl. *compiler*)

Prevođenje u izvršni kod (asembler, povezivanje, punjenje).

- Asemblerski kod je mnemonička, čovjeku čitljiva inačica izvornog programa gdje se koriste simboli umjesto memorijskih lokacija (iz tablice simbola) i operacija (kodna tablica). Asemblerski kod se zatim pomoću programa asemblera pretvara u objektni strojni kod koji više nije čovjeku čitljiv.

- U fazi povezivanja (engl. *link*) spaja se skup neovisno oblikovanih modula u obliku objektnog koda i datoteka knjižnica (engl. *library files*) u jedinstveni izvršni program.

- Generirani strojni kod je **relokatibilan** te se u fazi punjenja (engl. *load*) određuju apsolutne memorijske adrese (osim statičkih apsolutnih adresa ulazno-izlaznih naprava i sličnih entiteta) i program se pokreće u memoriji.

Ostale značajne metode formalne specifikacije i sinteze

- **TLA i TLA$^+$ (Temporal Logic of Actions)** (+ alat TLA Toolbox) – **L. Lamport (Microsoft)**
  - https://lamport.azurewebsites.net/tla/tla.html
- **Alloy – D. Jackson (MIT)**
  - http://alloytools.org/
- **ASM (Abstract State Machines) – Y. Gurevich (Microsoft)**
  - http://web.eecs.umich.edu/gasm/
  - http://research.microsoft.com/en-us/projects/asml/
- **VDM i VDM++** (*Vienna Development Method*) (+ alat Overture)
  - http://overturetool.org/
- **SDL - Specification and Description Language**
  - http://www.sdl-forum.org/

Formalna verifikacija - FV

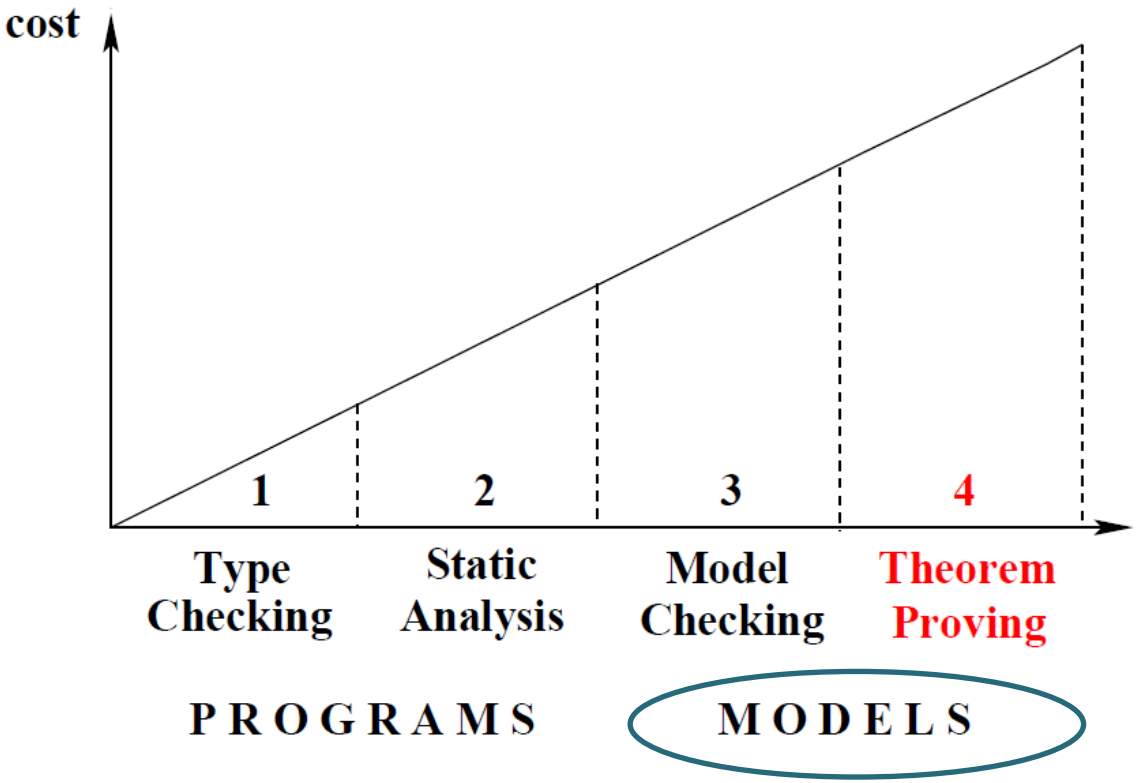
- **Definicija:** FV je proces analize sustava kojim dokazujemo da **implementacija** sustava posjeduje svojstva koju zahtijeva formalna **specifikacija** sustava

- "*Program testing can be used to show the presence of bugs, but never to show their absence.*" [Dijkstra, 1972]

- Nadamo se da "bug" ne postoji (nije nam poznato), ali želimo biti sigurni. Provodi se dokazivanje ili opovrgavanje ispravnosti algoritama u odnosu na zadanu formalnu specifikaciju ili svojstvo.
- Prolazi se svim putevima kroz program, a ne samo nekima kao kod ispitivanja.

Formalna verifikacija - FV

- FV provodi se u najvećem broju slučajeva nad **modelima** stvarnih programa (modelima implementacije)



Formalna verifikacija - FV

Metode formalne verifikacije

Provjera modela (engl. *Model checking*):

Provjerava zadovoljava li model implementacije zadano obilježje (specifikaciju).

Dokazivanje teorema (engl. *Theorem proving*):

Dokazuje u nekom logičkom formalizmu je li neka specifikacija logička posljedica implementacije (skupa formula).

Provjera ekvivalentnosti (engl. *Equivalence checking*):

Uspoređuje novo oblikovani model ili implementaciju s izvornim modelom ili implementacijom da bi se ustanovila logička ekvivalentnost.

Provjera tvrdnje (engl. *Assertion-based verification*):

Provjerava se da neko specifično svojstvo ili uvjet mora uvijek biti zadovoljeno.

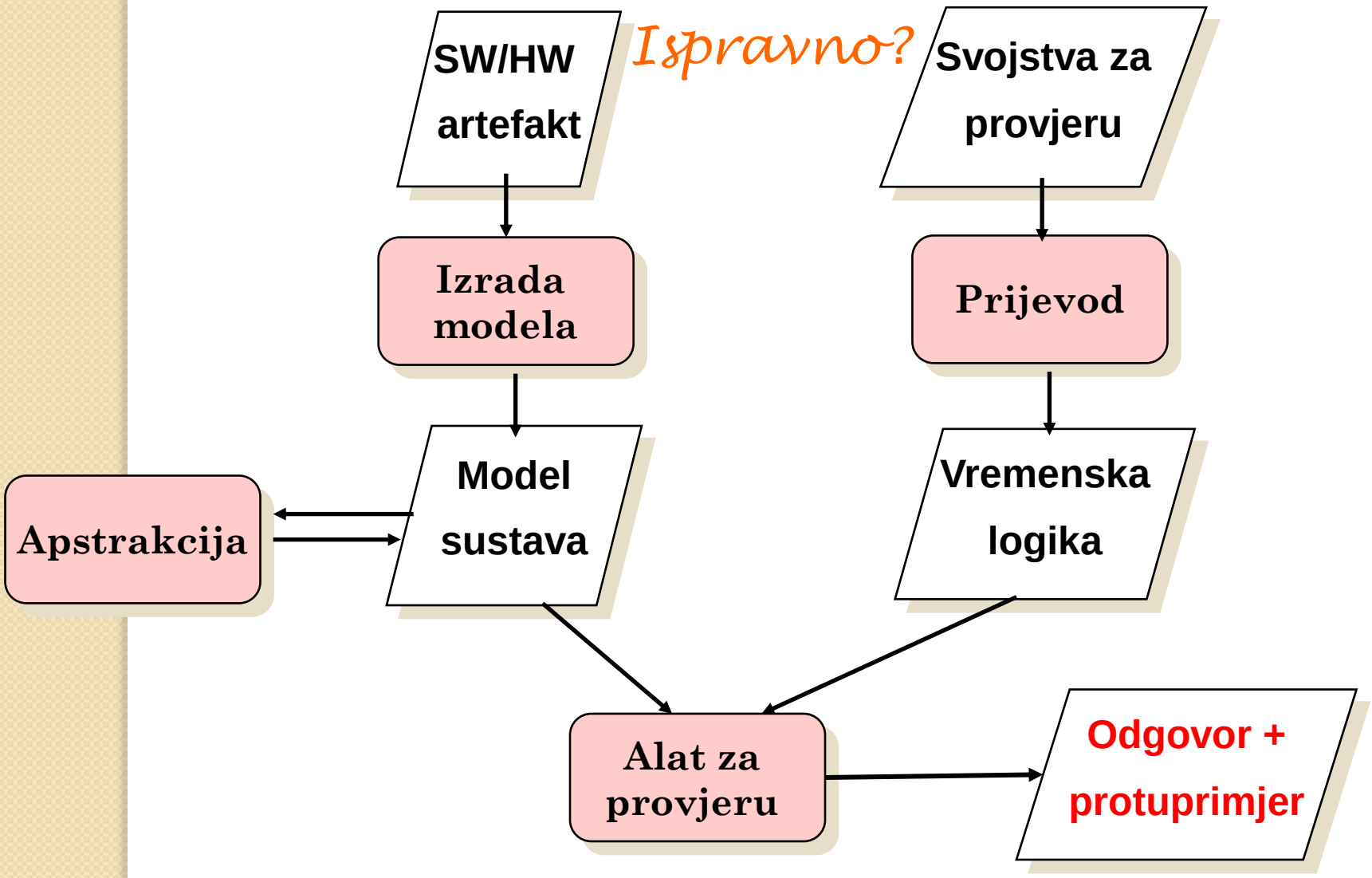
Provjera modela

- **Provjera modela** – automatizirana metoda provjere reaktivnih sustava modeliranih strojevima s konačnim brojem stanja na zadanu specifikaciju (obilježje, ponašanje).
- Provjerava se zadovoljava li u logičkom smislu (model) implementacije zadano obilježje (specifikaciju)
- Temelji se na potpunom pretraživanju prostora stanja (niti jedno stanje ne ostaje neispitano – razlika prema ispitivanju).
- Moguć vrlo veliki prostor stanja ($> 10^{150}$).

Provjera modela

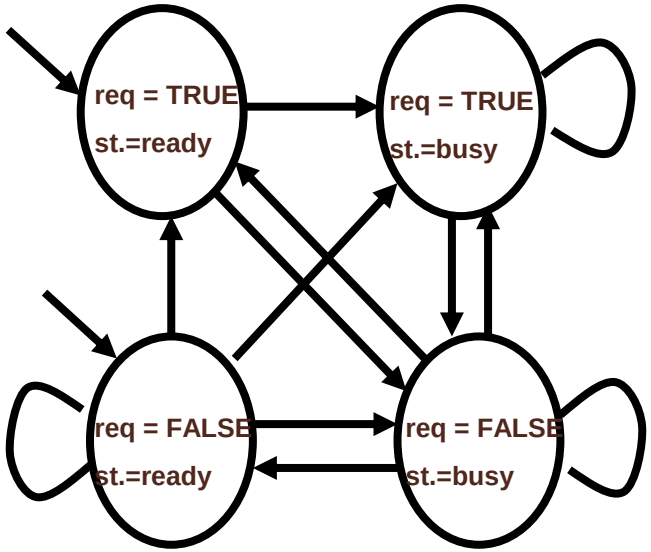
- **Proces provjere modela:**
  - Implementaciju prikazati modelom u formalizmu svojstvenom alatu za provjeru
  - Specifikaciju ponašanja prikazati nekom **logikom**, najčešće vremenskom logikom
  - Verifikacija je **automatizirana** danim alatom.
- Varijante:
  - **Ograničena provjera modela** (engl. *bounded model checking*) – provjera (modela) implementacije do nekog *k*-tog stanja
  - Provjera modela za sustave za rad u stvarnom vremenu – najčešće vremenski automati, vidjeti npr. Uppaal: http://www.uppaal.org/

FV: Provjera modela



FV: Provjera modela

Primjer modela implementacije opisanog u alatu za FV (NuSMV) – vrijedi samo za taj konkretni alat




```
MODULE main
VAR
  request:   boolean
  status:    {ready, busy}
ASSIGN
  init(status) := ready;
  next (status) := case
                   request: busy;
                   TRUE: {ready, busy}
                   esac;
```

Jedna formalna specifikacija u vremenskoj logici:

AG(request ⇒ AF (status = busy))

FV: Provjera modela

2007 Turing Award Winners Announced for their groundbreaking work on Model Checking

Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis are the recipients of the 2007 A.M. Turing Award for their work on an automated method for finding design errors in computer hardware and software. The method, called **Model Checking**, is the most widely used technique for detecting and diagnosing errors in complex **hardware and software** design. It has helped to improve the reliability of complex computer chips, systems and networks.

FV: Dokazivanje teorema

- Formalna verifikacija programa – provjera da program doista radi ono što bi trebao
- **Dokazivanje** rada programa – obrazloženje o programu kako bi se zajamčila točnost
- Dokazivanje rada programa ima smisla samo ako imamo neku vrstu **specifikacije** onoga što bi program trebao raditi
- Mnoge su specifikacije napisane na prirodnom jeziku što može dovesti do nepreciznosti i nesporazuma

- Za stvarne programe često se koristi **oblikovanje prema ugovoru** (engl. *design by contract*)
- Specifikacije su zadane kao **preduvjeti** i **postuvjeti** te **algebarska specifikacija invarijanti**

FV: Dokazivanje teorema

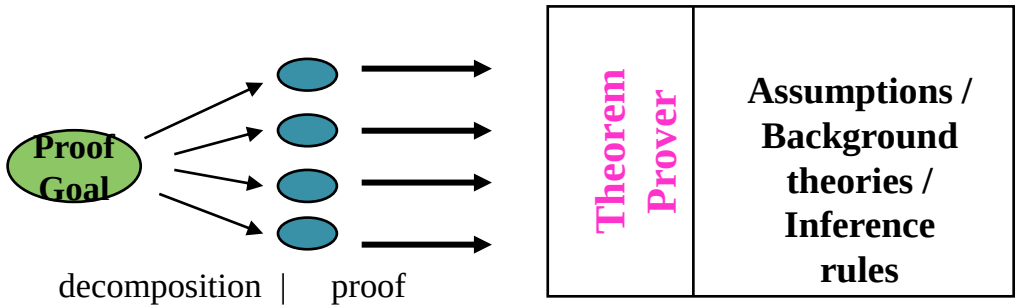
Zadatak dokazivanja teorema:

Dokazati da je neka **specifikacija logička posljedica implementacije** u nekom odabranom logičkom formalizmu.

Ulazi u sustav za dokazivanje teorema:

- Logičke formule koje opisuju implementaciju (tzv. aksiomi).
- Logička formula koja opisuje specifikaciju
- Pretpostavke o domeni problema (Npr. Vcc=True).
- Teorija (pravila dokazivanja, ranije dokazani teoremi, …).

Izlaz sustava: Dokaz da je **impl ⊨ spec**



FV: Dokazivanje teorema

- Metodu nije jednostavno primijeniti u industriji jer većina inženjera nema znanja o formalnoj matematičkoj logici.
- Automatizacija je ograničena, postupak često traži ručno vođenje.
- Potrebno duboko poznavanje alata za verifikaciju.

Primjer:

Predikatna logika u verifikaciji programa dokazivanjem teorema:

Stanje programa: PSTATE(tekuća_naredba, lista_varijabli)

Svaka naredba se mora preslikati u logiku. Npr. ASSIGN:

ASSIGN(xinstr, zassignto, wexpr, yinstr)

xinstr – tekuća naredba, zassignto – varijabla u koju se upisuje,

wexpr – izraz evaluacije prije zapisivanja, yinstr – sljedeća naredba

Pravila, npr. "Program u stanju 1 naredbom ASSIGN prelazi u stanje 2":

$(\text{PSTATE\_1} \wedge \text{ASSIGN}) \Rightarrow \text{PSTATE\_2}$

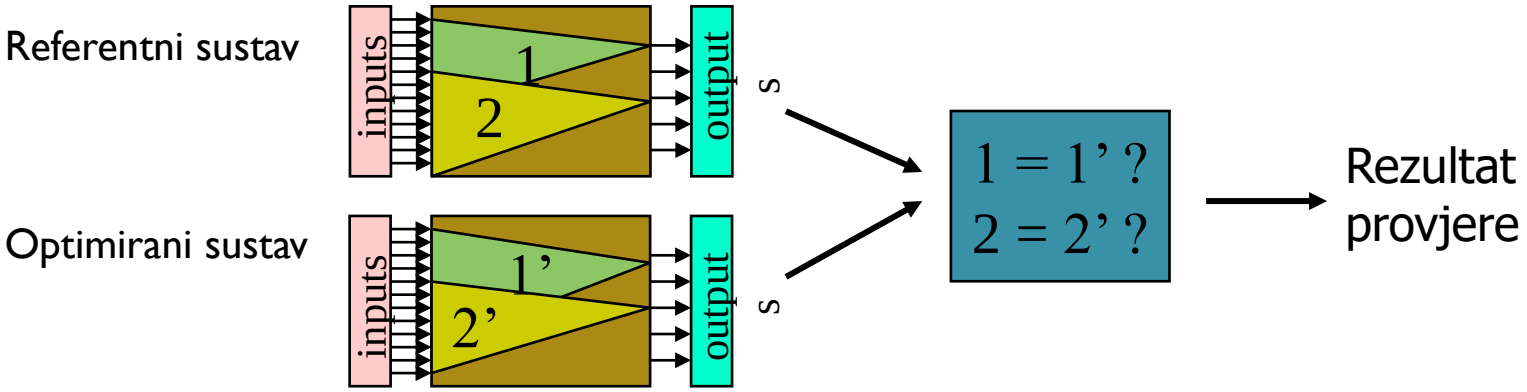
FV: Dokazivanje teorema

- Postoje mnogi razvijeni jezici i alati za dokazivanje teorema, primjerice:
  - **KeY-Hoare** – dokazivanje programa specifikacijama u Hoareovoj logici
    - http://i12www.ira.uka.de/key/download/
  - **Dafny** – programski jezik za pisanje sigurnih programa
    - https://marketplace.visualstudio.com/items?itemName=dafny-lang.ide-vscode
    - https://www.microsoft.com/en-us/research/wp-content/uploads/2016/12/krml220.pdf
  - **Verifast** – dokazivanje C i Java programa – separacijska logika
    - https://people.cs.kuleuven.be/~bart.jacobs/verifast/
  - **Coq** – dokazivanje različitih matematičkih svojstava – metoda računanja induktivnih konstrukcija = logika višeg reda + funkcijski jezik
    - https://coq.inria.fr/
  - **Spark2014** – formalna verifikacija programa pisanih u jeziku Ada 2012
    - https://www.adacore.com/about-spark
  - **Isabelle/HOL** – iskazivanje matematičkih formula u formalizmu i dokazivanje formula – logika višeg reda
    - https://www.cl.cam.ac.uk/research/hvg/Isabelle/index.html
  - …

FV: Provjera ekvivalentnosti

Kombinacijska provjera

Provjerava funkcijsku ekvivalentnost sustava koji nemaju memoriju (stanja).



Sustav se razbije na "logičke konuse" i uspoređuju se izlazi za jednake ulaze.

Značajno za sklopovlje pa nećemo obrađivati detaljnije u ovom predmetu.

FV: Provjera tvrdnje

- Opisuje se očekivani ili neočekivani uvjet (tvrdnja – "assertion") koji mora biti zadovoljen u implementaciji
- Tvrdnju se često uključuje u sustav koji ispitujemo
- Tvrdnja se najčešće opisuje proširenjima u jeziku implementacije, najčešće naredbom `assert(neki_uvjet)` koja mora biti ispunjena ili baca pogrešku
- Provjera tvrdnje se obično treba posebno uključiti prilikom pokretanja

```
// Java program to demonstrate syntax of assertion

class Test
{
    public static void main( String args[] )
    {
        int value = 15;
        assert value >= 20 : "Underweight";
        System.out.println("value is "+value);
    }
}
```

Zaključci o formalnim metodama (FM)

- Pojam ispravnosti (engl. *correctness*) je temeljni intelektualni izazov u industriji složenih sklopovskih i programskih proizvoda.
- Formalna verifikacija je rigorozna (matematički utemeljena) demonstracija ispravnosti.
- Dok je ručna verifikacija dostatna za male kritične sustave, složeni sustavi traže automatizirane postupke FV.
- Automatizirani postupci FV počivaju na skupu efikasnih alata (laganih za primjenu i brzih u donošenju odluke).
- Alati za FV potiču još brži razvitak gospodarstva temeljenog na složenim sklopovskim i programskim proizvodima.